Search Results [PDF FULL-TEXT 551 KB] PREV NEXT DOWNLOAD CITATION

On-chip repair and an ATE independent fusing methodology

Cowan, B. Farnsworth, O. Jakobsen, P. Oakland, S. **Ouellette, M.R.** Wheater, D.I
IBM, Essex Junction, VT, USA

This paper appears in: **Test Conference, 2002. Proceedings. Internation**

Publication Date: 7-10 Oct. 2002
On page(s): 178 - 186
ISSN: 1089-3539
Number of Pages: xvi+1250
Inspec Accession Number: 7528786

Abstract:
This paper describes a novel on chip repair system designed for ATE indepenc application on many unique very dense ASIC devices in a high turnover envir During test, the system controls on chip built-in self-test (BIST) engines, colle compresses repair data, programs fuses and finally decompresses and reload data for post fuse testing. In end use applications this system decompresses the repair data at power-up or at the request of the system.

Index Terms:
application specific integrated circuits automatic test equipment built-in self test elect integrated circuit testing maintenance engineering ASIC on-chip repairs ATE indepe methodology BIST built-in self-test engines e-fuse programming electrically prograr high turnover environment on chip repair systems post fuse testing repair data collection/compression repair data decompression/reloading unique very dense ASIC

Documents that cite this document
There are no citing documents available in IEEE Xplore at this time.

Search Results [PDF FULL-TEXT 551 KB] PREV NEXT DOWNLOAD CITATION

Home | Log-out | Journals | Conference Proceedings | Standards | Search by Author | Basic Search | Advanced Search | Join IEEE | Web Account | New this week | OPAC Linking Information | Your Feedback | Technical Support | Email Alerting | No Robots Please | Release Notes | IEEE Online Publications | Help | FAQ | Terms | Back to Top

On-Chip Repair and an ATE Independent Fusing Methodology

Bruce Cowan, Owen Farnsworth, Peter Jakobsen, Steve Oakland, Michael R. Ouellette, Donald L. Wheater,
IBM, Essex Junction Vermont

Abstract

This paper describes a novel on chip repair system designed for ATE independent application on many unique very dense ASIC devices in a high turnover environment. During test, the system will control on chip built-in self-test (BIST) engines, collect and compress repair data, program fuses and finally decompress and reload the repair data for post fuse testing. In end use application this system decompresses and loads the repair data at power-up or at the request of the system.

Introduction

As technology advances, new process and design techniques extend our possibilities while simultaneously limiting the development and reuse of yesterday's technology. This is the case for fuses. Historically, metal fuses were used for the storage of customization data and repair solutions of defects on silicon chips. Metal fuses required little area relative to devices and could be mechanically programmed with a laser. Today the area required by metal fuses is significantly greater than the area consumed by the decreasing size of devices. Metal fuses cannot scale with device technology because of the size requirements needed to mechanically program fuses. Metal fuses also require a direct "line-of-sight" for laser access, which complicates the physical design methodology, since fuses may not be placed underneath overlying power busses.

There is an additional factor further complicating this dilemma: as devices decrease in size, it becomes possible to put more devices onto silicon, leading to smaller chips with larger memory arrays that require more fuses to repair defects. The area penalty of metal fuses is now more costly than ever. To further aggravate the problem, mechanical fuse programming requires that all customization and repair data be collected, and stored off-line after each test. Once all data has been collected it must be compiled into a single repair solution and translated into XY coordinates corresponding to the fuse locations on the chip.

The customization, test and repair of complex chips in an automated manufacturing test environment with many unique ASIC designs are challenging problems. Initially metal fuses were located in the individual macros requiring identification or repair. For repair, steps have been taken to isolate metal fuses into remote laser fuse bays. With the ability to store fuse data in a remote location, further steps were taken in both hardware design and test software to serially organize and compress the repair data for fuse storage [1]. At system power-on reset and prior to post-fuse memory test, the fuse data is then decompressed into the original repair data form with the aid of an on-chip decompression system. This has helped reduce the penalty for using metal fuses, but has not eliminated the problem. In addition, the fuse system needs to accommodate varying device designs and types and numbers of memories as well as many diverse ATE Tool types.

The development of an electrically programmed fuse, or e-fuse, has opened the door to many possibilities. The e-fuse is manufactured as a polysilicon link and is significantly smaller than the metal fuse. It can also shrink in size with device technology, as the process continues to develop, because the e-fuse has fewer mechanical dependencies. Because the e-fuse is electrically programmed, it is now possible is to repair the chip multiple times; a first repair at 1st pass wafer final test (WFT), a second repair at 2nd pass WFT, and a third repair during the final test of the packaged chip. A multiple repair capability provides leverage for testing and repairing memories across various temperatures.

In this paper we will discuss our use of the e-fuse to enable on-chip self repair as part of manufacturing test. On-chip self-repair has led to the development of new design requirements for all BIST engines responsible for finding memory defects and collecting repair data [2,3]. It has also led to a methodology for a tester to communicate with a system controller regardless of the uniqueness of each ASIC design. This methodology controls system initialization,

0-7803-7542-4/02 $17.00

collection of repair data, and a method of storing this data in fuses.

The e-fuse

The enabling mechanism for on-chip self repair is the electrically-programmed fuse (e-fuse). The e-fuse can best be viewed as an electronically programmed read-only bit. The e-fuse element consists of several components: the polysilicon fuse, a fuse latch, a program latch, a program FET and a look-ahead programming multiplexor (mux). Figure 1 shows the connectivity of these components.

The first and primary component is the fuse [4,5]. The fuse has two possible logic states. It can remain intact where its value is evaluated as a logical 'zero'. The other state is "programmed" and evaluates to a logical 'one'. A DC current pulse of 10 mA in amplitude and duration of 200 usec is required to program the fuse. This high current programs the fuse by dramatically increasing the resistance of the polysilicon link. This method of programming poses an additional challenge with respect to delivering the current necessary to program each fuse. However, the necessity of limiting the fuse programming current to a single fuse at a time is balanced by the need to program only a subset of the total number of fuses on any given chip. This presents an opportunity for a significant savings in fuse-programming time.

A fuse latch is associated with each fuse. The fuse latch serves three functions: 1) during fuse programming, all fuse latches are initialized to a logical 'zero', and then a single logical 'one' is shifted through, providing a sequence enabling mechanism to determine when to program each fuse; 2) when reading the fuse values, the fuse latch is used to sense and store the fuse value; and 3) the fuse latch has an additional data port, used to create a daisy chain between all fuses, which permits serial access to all fuses. Because of the high DC current requirements required to program a single fuse, a logical 'one' is shifted through a field of zeroes using the daisy chained state latches in order to program one fuse at a time.

The remaining components are functional only when programming the fuse. The desired state of the fuses is shifted into the program latches. When a program latch is loaded with a logical 'one', it selects the program FET and also disables the look-ahead programming mux. The look-ahead mux causes the fuse latch's shifted 'one' to skip over fuses which are not selected for programming, thereby saving fuse programming time. For programming to occur, the EFuseProg signal is enabled and a high voltage is placed across the fuse using the Fsource input and the resulting current is sunk through the program FET. Only when this high current path is enabled can the fuse be programmed. The program FET is enabled when the program latch is set and the shifted 'one' arrives at the fuse latch.

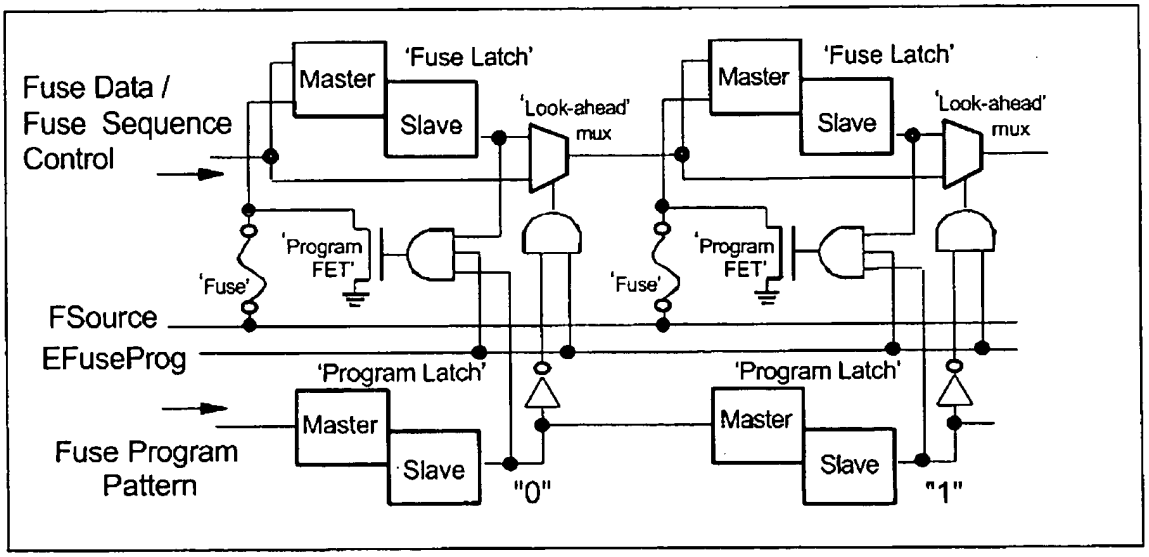

Figure 1. Diagram of (2) e-fuses

On Chip Repair using an e-fuse

The introduction of the e-fuse enables the opportunity for the development of on-chip repair, where at the conclusion of test the compressed repair data is programmed into e-fuses. This can be taken further to enable multiple repairs after varying test conditions or even after test of the packaged chip.

To enable multiple on-chip repairs, several new design requirements have to be defined. BIST engines must be capable of collecting a complete repair solution. The BIST engines must also be capable of adding additional new repairs to the previous repair solution. When not collecting repair data, the BIST engines must be disabled such that they do not modify the most recent repair data. Most importantly, the design needs to be able to perform many sequential operations without use of the scan chains, so that the state of all elements may be preserved for the next operation. For reduced pin-count logic testers to be used, the design also needs to avoid driving new test only I/O. Designs must be able to operate on any ATE via ATPG direct release mechanisms.

In order to perform on chip-repair, a complete redundancy solution has to be generated. This requires that all BIST engines be run prior to fuse programming. Before a BIST can run, it has to have access to previously stored results or be initialized so that it has a valid starting point. To add more complexity to the system not all BIST engines can operate concurrently. SRAM BIST engines may be run concurrently and DRAM BIST engines may be run concurrently, but SRAM and DRAM BIST engines may not be run concurrently. The SRAM and DRAM BIST engines have different operating requirements and different tester stimuli.

There are two new components needed to enable on-chip repair. A fuse controller (FUSECNTL) is needed to arbitrate control between the various BIST engines and perform the on system tasks needed to unload, load and program e-fuses. The second component needed for self repair is an e-fuse programmable shift register (PSR) macro. The e-fuse PSR macro contains many e-fuse elements as previously described in Figure 1 and multiple PSR macros will be used. Figure 2 shows the connectivity of these components.

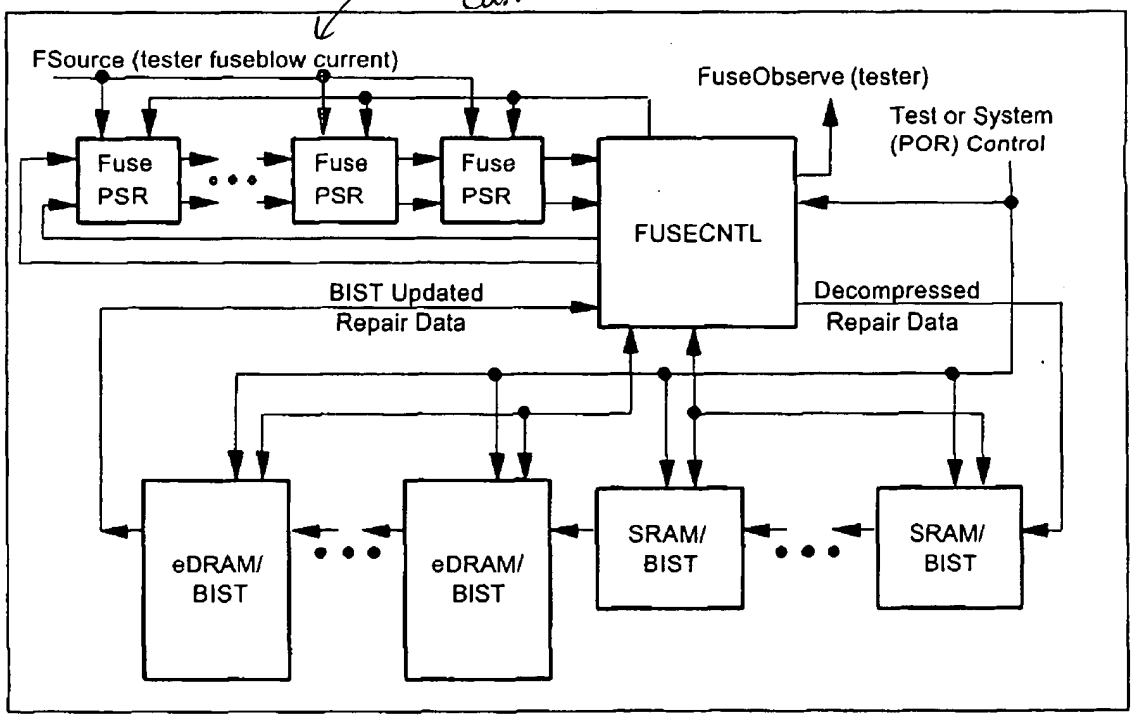

Figure 2. Diagram showing the on-chip repair system components

The fuse controller design is used to route BIST collected repair data from the repair register in the memories to the fuse PSRs prior to fuse programming. The controller is additionally used to route repair data from the fuse PSRs into the repair register of the memories prior to running BIST or during system memory operation.

The Fuse Bay

The fuse bay is used for storing the memories' repair solutions. Each fuse bay consists of a number of PSRs daisy chained together. The PSRs contain extra logic used for reading the fuses. Reading an unprogrammed fuse requires current to be sourced through the fuse, and when reading multiple fuses the required current sourced from the power supply rails may be substantial. Because of the power required to read fuses, only a sub-set of the fuses within one fuse PSR are read at one time. A sub-set of fuses is read when its read signal is asserted and latched. On the next clock cycle the read signal is transferred to the next sub-set of fuses.

As shown in Figures 3 and 4, the overall e-fuse system design approach actually contains three separate fuse bays; more specifically, the primary, secondary and tertiary fuse bays. The primary fusebay is used to program the first pass test's repair data, while the secondary and tertiary fuse bays are used to implement the second and third pass tests' repair data, respectively. Since most defects are found in the first pass of test, the primary fuse bay will be significantly larger than the secondary and tertiary fuse bays. The third pass of test will find the fewest defects, so that the tertiary fuse bay will be the smallest.

Each fuse bay is considered to contain valid repair data, if and only if, the first bit shifted out after a read, known as the 'fused' bit, is programmed to a logical 'one'. For the primary fuse bay, the next set of bits contain a count of the total number of latches in the repair register. This number is acquired and stored by the controller and is only programmed in the primary fusebay. The length of the repair register is needed the controller so that repair data may be properly shifted into and out of the repair register. The remaining fuses in the primary fuse bay contain the first pass test repair data. For the secondary and tertiary fuse bays, the bits following the 'fused' bit contain repair data.

The Controller

The controller creates a generic interface for the tester, independent of the design. Under normal system operation and during test, the controller is responsible for reading, shifting and decompressing the programmed repair data from the fuse bays into the memories' repair register. During test, the controller is additionally used to enable and control the appropriate BIST engines at the correct time, shift

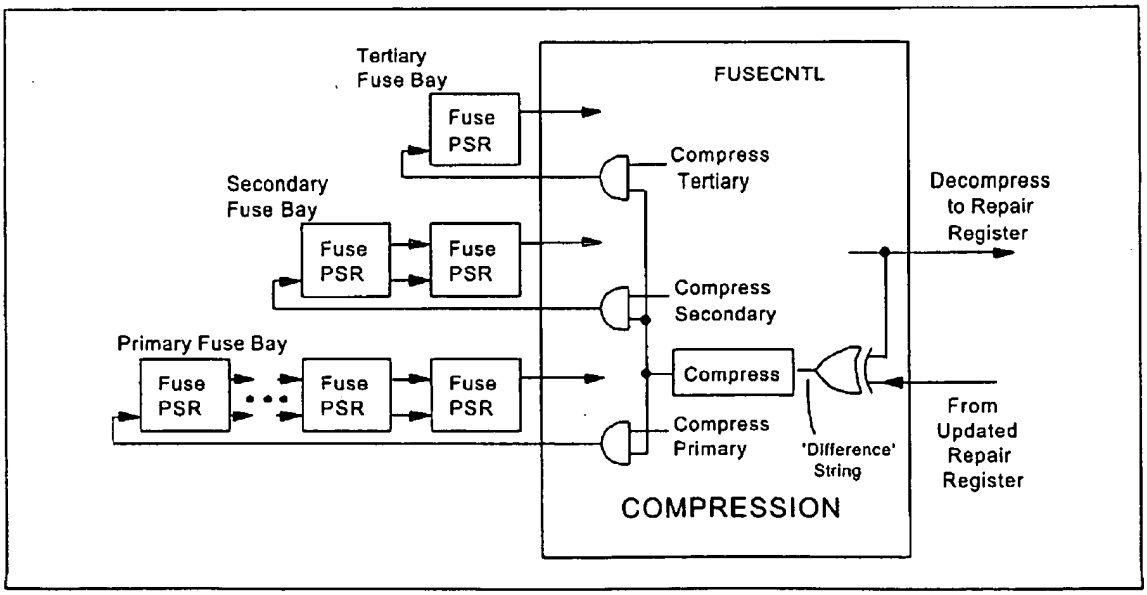

Figure 3. Compression into primary, secondary or tertiary fuse bay

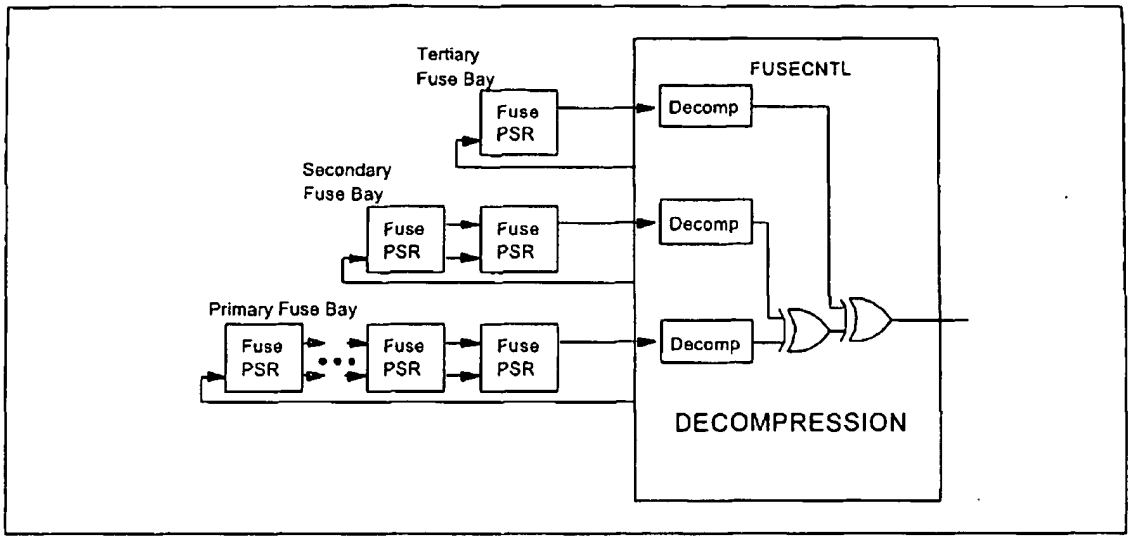

Figure 4. Decompression from primary, secondary and tertiary fuse bays

and compress the BIST updated repair data from the memories' repair register to the appropriate fuse bays, control fuse programming and verify that they are correctly programmed.

Each of these steps is controlled by the tester which provides the appropriate instruction to the controller through a serial access port. For each instruction issued to the controller, a status pin is made available by the controller for the tester to observe. This status pin is asserted at the successful completion of each instruction. If no assertion occurs after some predefined number of clock cycles, then an error occurred in that step or it is not possible to generate a repair solution for this chip. Based on the state of the status pin, the tester can respond appropriately by either continuing test or aborting and moving to the next chip and continuing test.

The controller is composed of two major components: an instruction processor and a repair data processing unit. The instruction processor controls the various steps by decoding instructions issued from the tester. The repair data processing unit contains the decompression and compression logic. Figure 3 shows the compression logic. Repair data from the memories' BIST-updated repair register is shifted through an exclusive-or gate where it is compared with the decompressed fuse data of the currently programmed fuse data. The resulting 'difference' serial string is compressed and shifted into the selected fuse bay for the appropriate level of repair. At first pass WFT (fuses have not yet been programmed) the currently programmed repair solution is all zeroes, so that the BIST-updated repair register data is unchanged by the exclusive-or gate. The unchanged repair register data is then compressed and shifted to the primary fuse bay. The serial compression algorithm will be discussed in detail later. Figure 4 shows the decompression logic. Fuse data from all three fuse bays is decompressed and combined into a single serial string, then shifted to the memories' repair register.

Controller at Test

The controller contains a serial-programmable instruction register that allows the tester to control the current step. These instructions include:

1. Get repair register length (design dependent)
2. Read fuse data
3. Decompress fuse data
4. BIST
5. Compress updated repair data
6. Transfer count value
7. Program fuses
8. Decompress/verify programmed fuses

At test, each step has to be explicitly enabled by the tester. In order for correct operation of compression and decompression, the decompression and compression logic needs to have access to the exact number of repair register latches used by the memories to store the repair data. The repair register length is different per chip design, depending how many of which types of memories are used. The "get

repair register length" instruction will count the design's repair register latches to provide this function. This instruction only needs to be executed during the first pass of test. The repair register length value is stored in the primary fuse bay as part of the first repair solution and is always available for future use. Prior to executing this command all repair register latches must be initialized to logical 'zeroes' during scan initialization. The count is generated by shifting a logical 'one' through the entire chain and incrementing a count for each clock cycle until the shifted 'one' is observed at the controller. At this point, the status pin is asserted instructing the tester that the instruction is complete.

Shift ng → Clock cycles

The "read fuses" instruction will cause the e-fuses to be read from all fuse bays. Because of the power demand associated with reading fuses, fuses are read by shifting a logical 'one' through the read latches in each fuse PSR, allowing only a sub-set of the fuses in the fuse bay to be sensed concurrently. When the read signal returns to the controller, all fuses have been read. The controller next proceeds to remove the fused bit from each fuse bay. If this bit is programmed, then the fuse bay contains fused data; otherwise it is available for programming. If the primary fuse bay is programmed, the repair register latch count is extracted. At this point, the next data available from the fuse bays is the compressed repair data, and so the shifting of fuse data from the fuse bays stops. When these operations have completed, the status pin is asserted.

The "decompress fuse data" instruction directs the controller to decompress the compressed repair data. If no data is available, the status pin is immediately asserted, instructing the tester to move on. In this case, the scan initialization serves as a valid starting point for the BIST engines. However, if there is data in the fuse bay, it is decompressed. This instruction contains one bit per fuse bay that can disable the decompression of data from that particular fuse bay. This capability allows a programmed fuse bay's data to be ignored during decompression and is available for system diagnostics. When decompression has successfully completed, the status pin is asserted and the tester can respond appropriately.

The "BIST" instruction releases control to the designated BIST as specified in the instruction. This instruction can enable either the DRAM or SRAM BIST. Each time this instruction is loaded, the specified BIST engine receives a restart signal to unlock it. When not in use, the BIST engine must disable itself so that it cannot corrupt its solution while being clocked. The BIST instruction also includes data bits that are used for specifying different modes of BIST operation. These bits give the tester the flexibility to vary the ways in which BIST is run; whether to collect failing addresses or set a pass/fail bit, vary the number of times BIST is run, and vary the order, without using the logic test scan chains. This allows a savings in test time when only SRAMs or DRAMs, but not both, are present on the chip design. When optimizing the test flow, a savings in test time may also be realized by running SRAM BIST prior to running DRAM BIST or vice-versa. At completion of running DRAM or SRAM BIST, the fuse controller's status pin allows the tester to observe whether the corresponding memories are repairable. Based on this information the tester can respond by either aborting and moving on to the next chip, or scanning data off line for analysis.

The "compress updated repair data" instruction performs several functions. After BIST is completed, the data in the repair register may have been updated. This updated repair data will be shifted into the fuse controller, while at the same time the fuse data from all three fuse bays' (data which was previously read with the read fuse data instruction) is shifted through the decompression logic. The output serial strings of the decompression logic and the incoming repair register data from the memories are compared with an exclusive-or gate to form a 'difference' string as shown in Figure 4. This difference string is then compressed and the compressed result is shifted into the selected fuse bay. For example, if a repair has already been programmed into the primary fuse bay, and a second level of repair is being performed, then the secondary fuse bay must be selected at this time. As the compressed data is being shifted, the fuse controller counts the number of logical 'ones' that are shifted into the program register of the selected fuse bays. Each logical 'one' represents a fuse that needs to be programmed. During compression, the repair data being shifted into the controller from the memories' repair register is also being shifted back into the memories' repair register, so that it may be later accessed to support diagnostics capabilities. At the completion of compression, the fuse controller's status pin is asserted and the logical 'ones' count value may be used by the tester to take full advantage of the e-fuse design in order to shorten the time required to program fuses.

The "transfer count value" instruction causes the controller to shift the logical 'ones' count value,

acquired during the compression instruction, to the tester via the fuse controller's status pin. The count value is of a pre-designated length. This completion of this instruction does not assert the status pin.

The "program fuses" instruction enables the programming of fuses as discussed earlier. A logical 'one' is shifted through the fuse latch register of the e-fuse PSRs to step through the programming of the e-fuses. When this logical 'one' returns to the controller, the status pin is once again asserted. Prior to the execution of this instruction, the tester must provide the necessary voltage on the Fsource pin (shown in Figure 1) for programming e-fuses. Likewise, at the completion of this instruction, the voltage should be disabled.

The "decompress/verify" instruction is used to decompress the newly programmed fuse data for setting up the final BIST test and at the same time verify that all fuses have been programmed correctly. Prior to running this instruction, all fuses must be read, using the previously described "read fuses" instruction. During the execution of the "decompress/verify" instruction, the fuse latch data and the program latch data are both shifted into the fuse controller, where they are compared. At the same time, the fuse data from all three fuse bays is decompressed. The previously BIST-updated repair register data is still available in the memories' repair register and is also being shifted into the fuse controller where it is being compared to the decompressed string. If any program latch/fuse latch or decompression/repair register miscompares occur, then either the fuses did not program correctly or the compression logic is not working properly. If no miscompares occur then the status pin is asserted at the completion of this instruction.

A flowchart describing the macro-test flow for testing and repairing memories using e-fuses is show in Figure 5. Each "box" in the flowchart represents a fuse controller instruction.

A set of latches is present in the fuse controller instruction register to select only one fuse bay during compression and to direct the shifted 'one' to the selected fuse bay's fuse latches during fuse programming. There is one instruction latch available per fuse bay and only one of these latches can be enabled per instruction. These latches are also logically combined with the first fuse of the fuse bay to determine whether or not to enable fuse programming in the selected fuse bay. If the 'fused' bit is programmed in the selected fuse bay then the attempted fuse programming operation is not allowed This protects the selected fuse bay from being programmed twice, as this could destroy the chip.

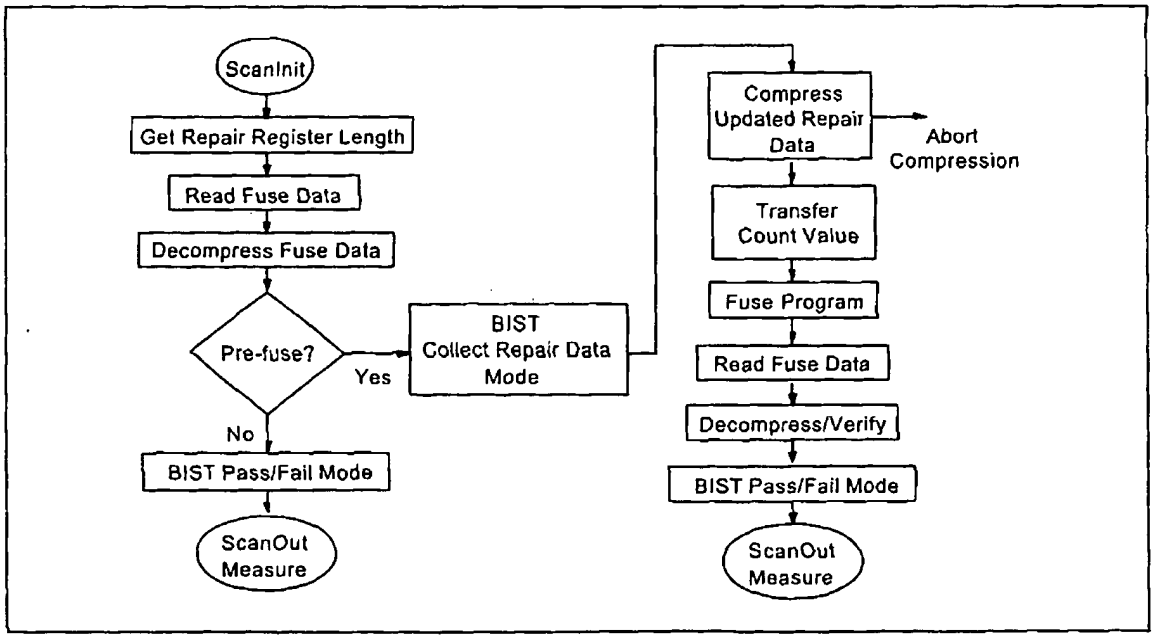

Figure 5. Test and Repair FlowChart

A master fuse program enable latch is also contained within the controller instruction register. This latch is reset during system power-on-reset, so that the fuse programming operation is prevented during normal system operation. If the master fuse program enable latch is not set during manufacturing test, only fuse data decompression will run, and upon successful completion, the controller will release control to both the SRAM and DRAM BIST engines.

Compression Algorithm

In order to better minimize the number of physical fuses over the previous design approach [1], an improved compression algorithm was required which uses a two bit opcode. The two bit opcode selects either an uncompressed mode or a runlength encoding of zeroes or ones, as shown in Figure 6. This variable-length instruction compression format is optimum for transforming a repair register's serial data into a compressed serial string.

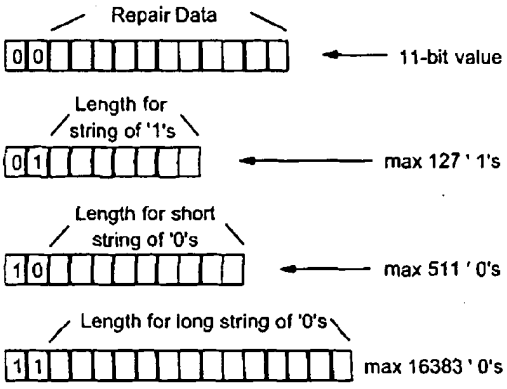

Figure 6. Compression Algorithm

Test Patterns and ATE Interface

The test patterns used to operate the fuse controller for the embedded memory test and repair process begin as generic macro-pattern rules (MPRs). The MPRs are qualified and released into the ASIC technology library. As the complete set of test patterns is created for each specific ASIC product, these generic MPRs are used by ATPG to automatically tailor and construct the test patterns for the on-chip e-fuse process.

The test patterns mirror the flow of the on-chip fuse controller and BIST application described in the previous sections. The test patterns and tester interface are simplified by the fact that the majority of the test and repair processes are controlled and applied by on-chip DFT.

The test patterns contain separate test sequences for each of the steps in the e-fuse process. The MPRs and ATPG insert identifying labels into each of these test sequences. These labels are used during the construction of the ATE test program.

Initialization

The initialization test sequence applies a scan, which initializes the controller, BIST engines, and memory repair registers appropriately. The remaining operations are each performed without using the scan chain access. The test sequence then issues the necessary instructions and clocks to prepare the BIST engines for test by decompressing any compressed fused data. After each instruction, the status pin is measured by the test sequence to verify correct operation of the instruction. A failure of this measure is used by the tester program to abort the pattern application and move on to the next chip or begin diagnostics.

Memory Tests

The initialization patterns are followed by a series of test sequences to control the BIST testing. Each test sequence causes the controller and BIST engines to apply an SRAM or DRAM test pattern. While the BIST engines run, on-chip registers associated with each repairable memory accumulate any fuse repair that is identified by the BIST patterns. While power is maintained to the product, the repair data is saved internally in the repair register. The accumulation registers and repair registers may be one and the same, depending upon how the BIST is designed. The tester control program may change test conditions between some of the test sequences. The memory tests can be applied at low, nominal, and high conductance corners. In addition, some of the test sequences contain labeled break points where the on-chip BIST and external tester control program will both pause for retention tests or, alternately, for the elevation of the voltage level to stress the product. Throughout these memory tests, the tester does not measure or accumulate any repair data. Thereby, the content of these test sequences is quite simple.

Each memory BIST test sequence starts by loading a new instruction into the fuse controller. The instruction indicates the specific BIST pattern to apply. The BIST engines are then enabled and clocked. Since memory elements are typically tested

in parallel, the duration clocking is long enough to test the largest memory to completion. As the smaller memories complete, they enter an idle or locked state. At the end of each BIST application, the controller sets the status pin appropriately. Each BIST test sequence ends by measuring this pin. A fail indicates that either the memory tested is not repairable or that the BIST did not complete for some reason. This gives the test program a chance to begin diagnostics or abort and move to the next chip.

Fuse Programming

The fuse program test sequence follows the application of the memory tests. This test sequence causes the controller shift the repair register data, run fuse compression and load the compressed repair data into the selected fuse bay. The count of fuses to be programmed, which is created by the controller, is read by the tester. This is used to minimize the time spent programming the fuses by modifying the repeat count for the 200us fuse program cycle. The modified pattern then raises the voltage on the Fsource pin sourcing current to program the fuse and applies the appropriate clocks for each fuse to be programmed. This is followed by a fuse read operation, and then a fuse date decompression operation with the compare function enabled. The fuse controller's status pin is then observed to verify that the fuses were programmed correctly.

Post Fuse Programming Test

After fuse programming, the memory repair actions may be verified. A test sequence runs fuse decompression and then the memory BIST test sequences are reapplied across the multiple test corners. At their conclusion, a test sequence uses a scan to measure the pass/fail bits for each of the memories. In the event that these results indicate that a memory has not been repaired, a decision can be made to either dispose of that chip or to re-test and apply any new repairs that may be found.

However, it is most probable that the optimum approach would be to reserve the second repair action for a second pass of WFT at a different temperature, so that temperature sensitive defects could be repaired. The third test and repair sequence would most likely be reserved for testing and repairing the final packaged chip.

Summary

We have shown an on-chip e-fuse system where memory test, repair data calculation, e-fuse programming and post fuse test all can occur in a single test contact without data ever leaving the chip. We've also shown how an additional fusing capability can allow for enhanced yield of packaged chips. This permits an efficient usage of e-fuses in a complex ASIC test environment.

References

[1] M. Ouellette, D Anand and P. Jakobsen, "Shared Fuse Macro for Multiple Embedded Memory Devices with Redundancy," *Proc. IEEE Custom Integrated Circuits Conference, pages 191-194. May 2001.*

[2] J. Dreibelbis, J. Barth Jr., R. Kho and H. Kalter, "Processor Based Built in Self Test for Embedded DRAM", *IEEE Journal of Solid State Circuits, pages 1731-1740, November 1998.*

[3] N. Watanabe, F. Morishita, Y. Taito, A. Yamazaki, T. Tanizaki, K. Dosaka, Y. Morooka, F. Igaue, K. Furue, Y. Nagura, T. Komoike, T. Morihara, A. Hachisuka, K. Arimoto and H. Ozaki, "An Embedded DRAM Hybrid Macro with Auto Signal Management and Enhanced-on-Chip Tester", *Proc. ISSCC Digest of Technical Papers, page 388, February 2001*

[4] C. Kothandaraman, Sundar K.Iyer, J.J. Wu and Subramanian S. Iyer, "Optimisation of $CoSi_2$ based electrical fuses for redundancy implementation in Sub-0.13um embedded DRAM applications", *Extended Abstracts of the 2000 International Conference on Solid State Device and Materials, Sendai 2000, pages 166-167 , August 2000.*

[5] V.Klee, J.Norum, R.Weaver, S.S.K.Iyer, C.R.Kothandaraman, J.Chiou, M.Chen, N.Kusaba, S.Lasserre, C.Liang, J.Liu, A.Lu, P.R.Parries, B.J.Park, J.Rice, N.Robson, D.Shum, B.Khan, Y.Liu, A.Sierkowski, C.Waskiewiscz, P.Wensley, T.Wu, J.Yan and S.S.Iyer, "A 0.13 um logic-based embedded DRAM technology with electrical fuses, Cu interconnect in SiLK™, sub-7ns random access time and its extension to the 0.10 mm generation", *International Electron Device Meeting 2001. Washington DC, pages 407-410, December 2001.*